Объекты синхронизации, продолжение

Операционные системы 2011/12

Татьяна Романова

10 октября 2011 г.

План на сегодня

- Мониторы и условные переменные (добавление в очередь)
- Проблема читателей и писателей
- Блокировки чтения/записи
- Барьеры
- Каналы

Материалы

- CS 162, лекция 7 (видео лекции на YouTube).
- Э. Таненбаум, Современные операционные системы, глава 2 (межпроцессное взаимодействие), глава 3.

Формальности

- 15 октября можно (и нужно!) с 15.10 сдавать лабораторные по ОС Семёну в 440Б (семинара у 08-206 не будет, лекцию проводим сегодня).
- ОС: все остальные семинары (29.10 у 08-206), лекции (22.10, 29.10, 05.11) и лабы (каждую субботу и понедельник) будут. Занятия будет вести Семён Чечеринда.
- 29 октября во второй части лекции будет колоквиум по пройденным темам (процессы, сигналы, потоки, синхронизация).
- Лабораторных по ДА 21 октября не будет. Следущая сдача лабораторных по ДА: 18 ноября.

Сложные объекты синхронизации

Мотивация

Все можно сделать с помощью семафоров, но

- для некоторых задач решение оказывается слишком сложным
- сложное решение часто содержит ошибки, его труднее понять и доказать корректность
- пример: в проблеме производителя и потребителя изменение порядка операций ведет к взаимоблокировке
- правильнее выделить некоторые шаблоны в отдельные объекты синхронизации

Условные переменные: побуждающий пример

- Синхронизированная работа нескольких потоков с очередью
- При удалении: спать, пока очередь пуста
- При добавлении: сообщать, что в очереди появился элемент
- Можно использовать семафор, инициализированный нулем?

```
if (queue.empty()) {
    sem.down(); // Ожидание события
}
queue.dequeue();
```

Условные переменные: побуждающий пример

- Синхронизированная работа нескольких потоков с очередью
- При удалении: спать, пока очередь пуста
- При добавлении: сообщать, что в очереди появился элемент
- Можно использовать семафор, инициализированный нулем?

```
if (queue.empty()) {
    sem.down(); // Ожидание события
}
queue.dequeue();
```

- Нельзя: проверка условия и удаление должны быть атомарны.

Условные переменные: определение

- Суть: ожидание выполнения некоторого условия.
- Поддерживают три операции:
  - wait(&lock) — ждет сингала внутри критической области
  - signal() — перемещает процесс, заблокированный на данной условной переменной, из очереди сна в очередь выполнения
  - broadcast() — перемещает все заблокированные процессы из очереди сна в очередь выполнения.
- В отличие от семафоров: может ждать внутри критической области (блокировка освобождается перед уходом в сон и снова захватывается после)
- Просто использовать, если не думать о реализации.
- Posix-реализация теряет сингнал, если его никто не ждет.

Мониторы

- Подход (парадигма), позволяющий гарантировать, что поток, вызывающий текущую процедуру не будет прерван кем-то еще.
- Встроены в некоторых языка (synchronized в Java), в других могут быть реализованы через использование мьютексов и условных переменных.
- Существуют два подхода к планированию после того, как поток, делающий signal покинет монитор:
  - Мониторы Хоара: после потока, вызывающего signal, сразу будет выполняться разбуженный поток. Сложная реализация, не используются на практике.
  - Мониторы Хансена: при вызове signal спящий поток становится готов к выполнению, но кто начнет выполняться на самом деле — решает планировщик. Используются на практике.

Реализация условных переменных

1. На семафорах:
   - в wait нужно отпустить блокировку, заблокироваться на семафоре, затем снова захватить блокировку.
   - в signal разблокировать семафор.
   - Не совсем корректна: сигнал должен теряться, когда никто не ждет.
2. В ядре:
   - wait:
     - добавить поток в очередь сна (операция с очередью обернута спин-блокировкой)
     - отпустить блокировку
     - switch()
     - захватить блокировку
   - signal:
     - удалить поток из очереди сна
     - добавить его в очередь на выполнение

Пример использования монитора: очередь

```
mutex_t m;
queue q;
cond_t c;

add_to_queue(item) {
    m.lock();
    q.enqueue(item);
    c.signal(); // Будим ждущего
    m.unlock();
}
```

```
del_from_queue() {
    m.lock();
    //Если пуста, заснуть
    while (q.empty())
        c.wait(&m);
    item = q.dequeue();
    m.unlock();
    return item;
}
```

Пример использования монитора: очередь

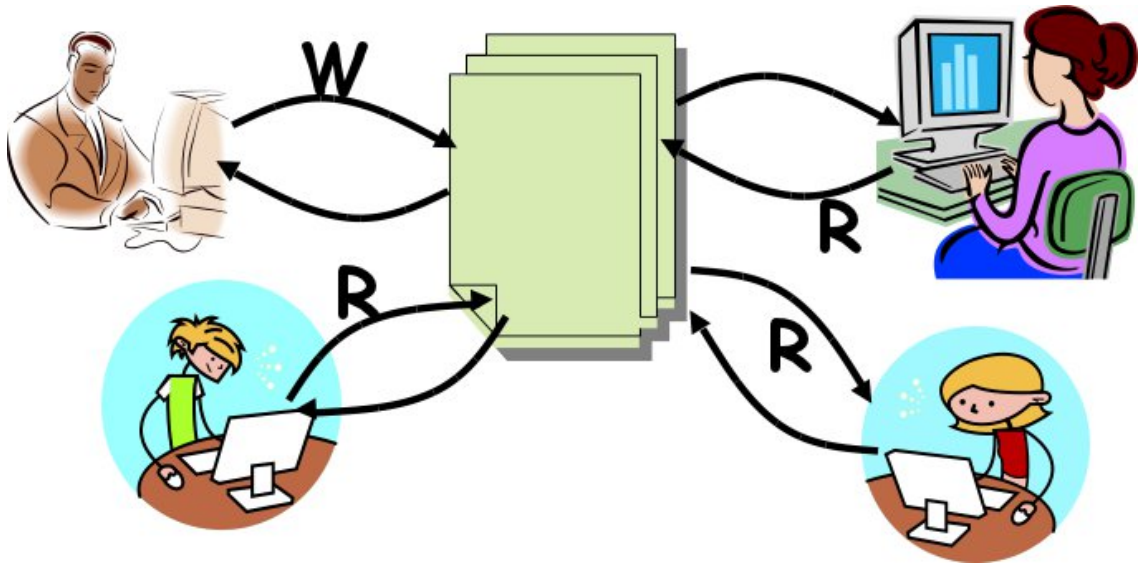
```
mutex_t m;
queue q;
cond_t c;

add_to_queue(item) {
    m.lock();
    q.enqueue(item);
    c.signal(); // Будим ждущего
    m.unlock();
}
```

```
del_from_queue() {
    m.lock();
    //Если пуста, заснуть
    while (q.empty())
        c.wait(&m);
    item = q.dequeue();
    m.unlock();
    return item;
}
```

Почему while, а не if?

Проблема читателей и писателей



- Существуют 2 класса пользователей:
  - Читатели читают и никогда не меняют базу
  - Писатели читают и меняют базу
- Блокировать всю базу при доступе каждого невыгодно.

Условия

- У читателей есть доступ, когда никто не пишет
- У пишсателей — когда никто не пишет и не читает
- Все изменения переменных состояния (сколько пишут и сколько читают) атомарны

Читатель

- Ждет, пока не будет писателей (работающих или ждущих)
- Работает с базой
- При выходе будит писателя (если он есть).

Писатель

- Ждет, пока не будет работающих читателей или писателей
- Работает с базой
- При выходе будит писателя или читателей.

Перемерные состояния

Переменные, гарантирующие корректную работу:

- AR: количество активных читателей
- WR: количество ожидающих читателей
- AW: количество активных писалетей
- WW: количество ожидающих писателей

Защищены мьютексом.

Условные переменные для передачи управления:

- ok_to_read
- ok_to_write

Реализация читателя

```
Reader() {
    mutex.lock();
    while (AW + WW > 0) {
        WR++;
        ok_to_read.wait(&mutex);
        WR--;
    }
    AR++;
    mutex.unlock();
    read();
    mutex.lock()
    AR--;
    if (AR == 0 && WW > 0)
        ok_to_write.signal();
    mutex.unlock();
}
```

Реализация писателя

```
Writer() {
    mutex.lock();
    while (AW + AR > 0) {
        WW++;
        ok_to_write.wait(&mutex);
        WW--;
    }
    AW++;
    mutex.unlock();
    write();
    mutex.lock()
    AW--;
    if (WW > 0)
        ok_to_write.signal();
    else if (WR > 0)
        ok_to_read.broadcast();
    mutex.unlock();
}
```

Обсуждение решения

- Как будет выполняться последовательность R1, R2, W1, R3?
- Честно ли делится время между писателями и читателями?
- Нужно ли условия в читателе перед тем как будить писателя?
- Можно ли использовать одну условную переменную вместо двух?

Поддержка в pthreads

- Условные переменные:
  - pthread_cond_wait принимает мьютекс, атомарно разблокирует его и помещает текущий поток в очередь ожидания. Поток просыпается после сигнала и блокирует мьтекс снова.
  - pthread_cond_signal переводит один заблокированный на этой переменной поток из сост. сна в сост. готовности
  - pthread_cond_broadcast: как signal, но для всех ждущих потоков
- Блокировки чтения/записи:
  - pthread_rwlock_rdlock, pthread_rwlock_unlock, pthread_rwlock_wrlock: писатели имеют приоритет.

Каналы

- Средство межпроцессного взаимодействия (используюется в Newsqueak, Alef, Go)
- Похожи на pipe, но по каналу можно передавать не только последовательность байт, но и произвольные объекты.
- Можно реализовать через ограниченный буфер: читающие ждут, когда канал пустой, пишущие ждут, когда канал заполнен.
- Интерфейс: init(n), read, write.

Барьеры

- Механизм полезен в основном для групп процессов.
- Приложение может делиться на фазы, новую фазу нельзя начинать, пока не закончится предыдущая
  - Пример: решение задач математической физики, несколько потоков могут работать с одной матрицей, но перейти к след. итерации должны все вместе.
- Интерфейс:
  - init(n), n — количество потоков, которые должны прийти к барьеру.
  - wait() — блокирует поток до тех пор, пока n потоков не придут к барьеру.

Заключение

Итоги лекции

- Для некоторых задач лучше подходят специальные объекты синхронизации.
- Условные переменные позволяют ждать наступления события.
- Мониторы являются принципом использования блокировок, встроенным в некоторых языках. Реализуются через услоные переменные и мьютексы.
- Блокировки чтения-записи позволяют различать доступ (совместный или исключительный) к ресурсам.
- Каналы позволяют обмениваться сообщениями через ограниченный буфер.
- Барьеры дают возможность начинать некоторое дейтсвие одновременно.